BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

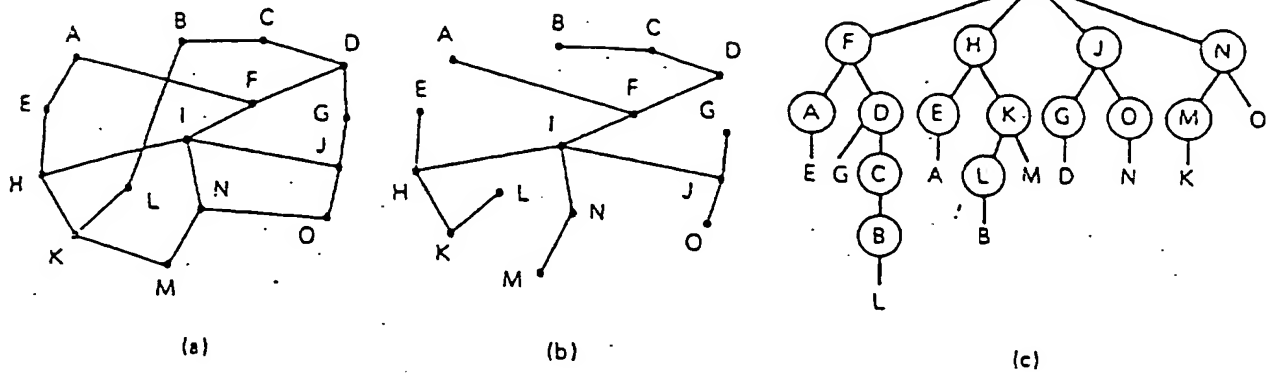

Fig. 5-21. Reverse path forwarding.

packet and a priori knowledge of the subnet diameter, or a list of packets already seen per source).

5.3. CONGESTION CONTROL ALGORITHMS

In this section we will examine five strategies for controlling congestion. These strategies involve allocating resources in advance, allowing packets to be discarded when they cannot be processed, restricting the number of packets in the subnet, using flow control to avoid congestion, and choking off input when the subnet is overloaded.

5.3.1. Preallocation of Buffers

If virtual circuits are used inside the subnet, it is possible to solve the congestion problem altogether, as follows. When a virtual circuit is set up, the call request packet wends its way through the subnet, making table entries as it goes. When it has arrived at the destination, the route to be followed by all subsequent data traffic has been determined and entries made in the routing tables of all the intermediate IMPs.

Normally, the call request packet does not reserve any buffer space in the intermediate IMPs, just table slots. However, a simple modification to the setup algorithm could have each call request packet reserve one or more data buffers as well. If a call request packet arrives at an IMP and all the buffers are already reserved, either another route must be found or a "busy signal" must be returned to the caller. Even if buffers are not reserved, some aspiring virtual circuits may have to be rerouted or rejected for lack of table space, so reserving buffers does not add any new problems that were not already there.

By permanently allocating buffers to each virtual circuit in each IMP, there will

always be a place to store any incoming packet until it can be forwarded. First consider the case of a stop-and-wait IMP-IMP protocol. One buffer per virtual circuit per IMP is sufficient for simplex circuits, and one for each direction is sufficient for full-duplex circuits. When a packet arrives, the acknowledgement is not sent back to the sending IMP until the packet has been forwarded. In effect, an acknowledgement means that the receiver not only received the packet correctly, but also that it has a free buffer and is willing to accept another one. If the IMP-IMP protocol allows multiple outstanding packets, each IMP will have to dedicate a full window's worth of buffers to each virtual circuit to completely eliminate the possibility of congestion.

When each virtual circuit passing through each IMP has a sufficient amount of buffer space dedicated to it, packet switching becomes quite similar to circuit switching. In both cases an involved setup procedure is required. In both cases substantial resources are permanently allocated to specific connections, whether or not there is any traffic. In both cases congestion is impossible because all the resources needed to process the traffic have already been reserved. And in both cases there is a potentially inefficient use of resources, because resources not being used by the connection to which they are allocated are nevertheless unavailable to anyone else.

Because dedicating a complete set of buffers to an idle virtual circuit is expensive, some subnets may use it only where low delay and high bandwidth are essential, for example on virtual circuits carrying digitized speech. For virtual circuits where low delay is not absolutely essential all the time, a reasonable strategy is to associate a timer with each buffer. If the buffer lays idle for too long, it is released, to be reacquired when the next packet arrives. Of course, acquiring a buffer might take a while, so packets will have to be forwarded without dedicated buffers until the chain of buffers can be set up again.

5.3.2. Packet Discarding

Our second congestion control mechanism is just the opposite of the first one. Instead of reserving all the buffers in advance, nothing is reserved in advance. If a packet arrives and there is no place to put it, the IMP simply discards it. If the subnet offers datagram service to the hosts, that is all there is to it: congestion is resolved by discarding packets at will. If the subnet offers virtual circuit service, a copy of the packet must be kept somewhere so that it can be retransmitted later: One possibility is for the IMP sending the discarded packet to keep timing out and retransmitting the packet until it is received. Another possibility is for the sending IMP to give up after a certain number of tries, and require the source IMP to time out and start all over again.

Discarding packets at will can be carried too far. It is clearly stupid in the extreme to ignore an incoming packet containing an acknowledgement from a neighboring IMP. That acknowledgement would allow the IMP to abandon a by-

now-received packet and thus free up a buffer. However, if the IMP has no spare buffers, it cannot acquire any more incoming packets to see if they contain acknowledgements. The solution is to permanently reserve one buffer per input line to allow all incoming packets to be inspected. It is quite legitimate for an IMP to examine a newly arrived packet, make use of any piggybacked acknowledgement, and then discard the packet anyway. Alternatively, the bearer of good tidings could be rewarded by keeping it, using the just freed buffer as the new input buffer.

If congestion is to be avoided by discarding packets, a rule is needed to tell when to keep a packet and when to discard it. Irland (1978) studied this problem and came up with a simple, yet effective heuristic for discarding packets. In the absence of any explicit rule to the contrary, a single output line might hog all the available buffers in an IMP, since they are simply assigned first come, first served. Figure 5-22(a) shows an IMP with a total of 10 buffers. Three of these are permanently assigned to the input lines. The remaining seven are holding packets queued for transmission on one of the output lines.

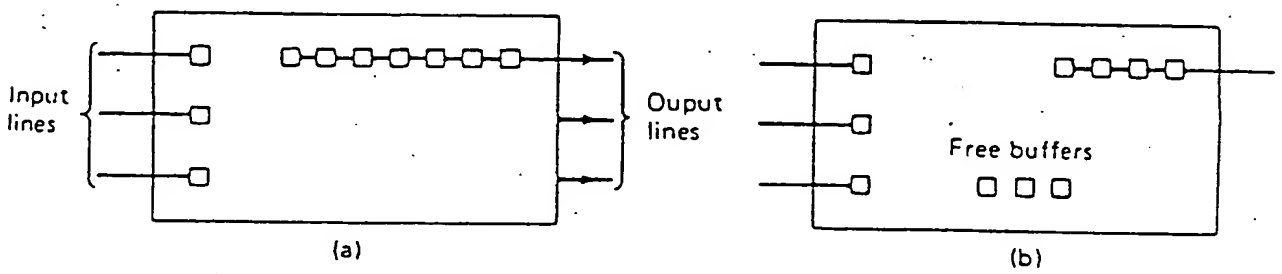

Fig. 5-22. Congestion can be reduced by putting an upper bound on the number of buffers queued on an output line.

Even though two output lines are idle, any incoming packets destined for these lines must be discarded because there are no spare buffers. This is obviously wasteful. Irland's idea is to limit the number of buffers that may be attached to any one output queue. For example, if the limit were set at four, the situation of Fig. 5-22(b) would prevail: three unassigned buffers. A newly arrived packet wanting to go out on the first output line would be discarded rather than allowing it to increase the queue length to five.

This strategy is not really as drastic as it may appear. After all, that output line is already running at maximum capacity. Having seven packets queued instead of four will not pump the bits out any faster, but it will allow traffic for the other lines to be forwarded immediately, possibly doubling or tripling the output rate of the IMP. In any case, the discarded packet will be retransmitted shortly. If the system is well tuned, it will even be retransmitted before the queue empties, so its initial rejection will not even be noticed.

Irland studied several different algorithms for determining maximum queue length, m, for an IMP with k buffers (buffers permanently dedicated for input do not count). The uncontrolled case is $m = k$. If there are s output lines, the case $m = k/s$ effectively means that each buffer is dedicated to a given output line. No

line may borrow even one buffer from an idle line, ever. Intuitively this is not efficient, and the study bears this out.

It turns out that the optimal value of m is a complicated function of the mean traffic. Although the IMP could attempt to measure its traffic and continually adjust m, if the traffic were bursty, this probably would not work well. Irland did, however, discover a simple rule of thumb that usually gives good, but not optimal, performance: $m = k/\sqrt{s}$. For example, for seven pool buffers and three lines, $m = 7/\sqrt{3}$, so 4 buffers would be allocated.

A related idea, due to Kamoun (1976), directly prevents any line or lines from starving: a minimum number of buffers is dedicated to each line. If there is no traffic, the empty buffers are reserved. Irland's method can be combined with Kamoun's by having a minimum and a maximum number of buffers for each line. The ARPANET uses this method.

Although discarding packets is easy, it has some disadvantages. Chief among these is the extra bandwidth needed for the duplicates. If the probability of a packet being discarded is p, the expected number of transmissions before it is accepted is $1/(1-p)$. A related issue is how long the timeout interval should be. If it is too short, duplicates will be generated when they are not needed, making the congestion worse. If it is too long, the delay will suffer.

One way to minimize the amount of bandwidth wasted on the retransmission of discarded packets is to systematically discard packets that have not yet traveled far and hence do not represent a large investment in resources. The limiting case of this strategy is to discard newly arrived packets from hosts in preference to discarding transit traffic. For example, IMPs could refuse or discard new packets from attached hosts whenever the number of buffers tied up by new packets (or total packets) exceeds some threshold.

5.3.3. Isarithmic Congestion Control

Congestion occurs when there are too many packets in the subnet. A direct approach to controlling it is to limit the number of packets in the subnet. Davies (1972) proposed a method that enforces precisely such a limit.

In this method, called isarithmic because it keeps the number of packets constant, there exist permits, which circulate about within the subnet. Whenever an IMP wants to send a packet just given to it by its host, it must first capture a permit and destroy it. When the destination IMP finally removes the packet from the subnet, it regenerates the permit. These simple rules ensure that the number of packets in the subnet will never exceed the number of permits initially present.

However, this method has some problems. First, although it does guarantee that the subnet as a whole will never become congested, it does not guarantee that a given IMP will not suddenly be swamped with packets.

Second, how to distribute the permits is far from obvious. To prevent a newly generated packet from suffering a long delay while the local IMP tries to scout up a

permit, the permits must be uniformly distributed, so that every IMP has some. On the other hand, to permit high-bandwidth file transfer, it is undesirable for the sending IMP to have to go hunting all over the place to find enough permits. It would be nicer if they were all centralized, so that requests for substantial numbers could be honored quickly. Some compromise must be found, such as having a maximum number of permits that may be present at any IMP, with excess permits required to hunt for an IMP with space. Note that the random walk of the excess permits itself puts a load on the subnet.

Third, and by no means least, if permits ever get destroyed for any reason, (e.g., transmission errors, malfunctioning IMPs, being discarded by a congested IMP), the carrying capacity of the network will be forever reduced. There is no easy way to find out how many permits still exist while the network is running.

5.3.4. Flow Control

Some networks (notably the ARPANET) have attempted to use flow control mechanisms to eliminate congestion. Although flow control schemes can be used by the transport layer to keep one host from saturating another, and flow control schemes can be used to prevent one IMP from saturating its neighbors, it is difficult to control the total amount of traffic in the network using end-to-end flow control rules. Still, if the hosts are forced to stop transmitting due to strict flow control rules, the subnet will not be as heavily loaded.

Flow control cannot really solve congestion problems for a good reason: computer traffic is bursty. Most of the time an interactive user sits at his terminal scratching his head, but once in a while he may want to scan a large file. The potential peak traffic is vastly higher than the mean rate. Any flow control scheme which is adjusted so as to restrict each user to the mean rate will provide bad service when the user wants a burst of traffic. On the other hand, if the flow control limit is set high enough to permit the peak traffic to get through, it has little value as congestion control when several users demand the peak at once. (If half the people in the world suddenly picked up their telephones to call the other half, there would be a lot of busy signals; the telephone system is also designed for average traffic, not worst case.)

When flow control is used in an attempt to quench congestion, it can apply to the traffic between pairs of:

1. User processes (e.g., one outstanding message per virtual circuit).
2. Hosts, irrespective of the number of virtual circuits open.
3. Source and destination IMPs, without regard to hosts.

In addition, the number of virtual circuits open can be restricted.

5.3.5. Choke Packets

Although limiting the volume of traffic between each pair of IMPs or hosts may indirectly alleviate congestion it does so at the price of potentially reducing throughput even when there is no threat of congestion. What is really needed is a mechanism that is triggered only when the system is congested.

One way is to have each IMP monitor the percent utilization of each of its output lines. Associated with each line is a real variable, u, whose value, between 0.0 and 1.0, reflects the recent utilization of that line. To maintain a good estimate of u, a sample of the instantaneous line utilization, f (either 0 or 1), can be made periodically and u updated according to

$$u_{new} = au_{old} + (1 - a)f$$

where the constant a determines how fast the IMP forgets recent history.

Whenever u moves above the threshold, the output line enters a "warning" state. Each newly arriving packet is checked to see if its output line is in warning state. If so, the IMP sends a choke packet back to the source host, giving it the destination found in the packet. The packet itself is tagged (a header bit is turned on) so that it will not generate any more choke packets later, and is forwarded in the usual way.

When the source host gets the choke packet, it is required to reduce the traffic sent to the specified destination by X percent. Since other packets aimed at the same destination are probably already under way and will generate yet more choke packets, the host should ignore choke packets referring to that destination for a fixed time interval. After that period has expired, the host listens for more choke packets for another interval. If one arrives, the line is still congested, so the host reduces the flow still more and begins ignoring choke packets again. If no choke packets arrive during the listening period, the host may increase the flow again. The feedback implicit in this protocol should prevent congestion, yet not throttle any flow unless trouble occurs.

Several variations on this congestion control algorithm have been proposed. For one, the IMPs could maintain two critical levels. Above the first level, choke packets are sent back. Above the second, incoming traffic is just discarded, the theory being that the host has probably been warned already. Without extensive tables it is difficult for the IMP to know which hosts have been warned recently about which destinations, and which hosts have not.

Another variation is to use queue lengths instead of line utilization as the trigger signal. The same exponential weighting can be used with this metric as with u, of course. Yet another possibility is to have the IMPs propagate congestion information along with routing information, so that the trigger is not based on only one IMPs observations, but on the fact that somewhere along the path there is a bottleneck. By propagating congestion information around the subnet, choke

packets can be sent earlier, before too many more packets are under way, thus preventing congestion from building up.

5.3.6. Deadlocks

The ultimate congestion is a **deadlock**, also called a **lockup**. The first IMP cannot proceed until the second IMP does something, and the second IMP cannot proceed because it is waiting for the first IMP to do something. Both IMPs have ground to a complete halt and will stay that way forever. Deadlocks are not considered a desirable property to have in your network.

The simplest lockup can happen with two IMPs. Suppose that IMP *A* has five buffers, all of which are queued for output to IMP *B*. Similarly, IMP *B* has five buffers, all of which are occupied by packets needing to go to IMP *A* [see Fig. 5-23(a)]. Neither IMP can accept any incoming packets from the other. They are both stuck. This situation is called **direct store-and-forward lockup**. The same thing can happen on a larger scale, as shown in Fig. 5-23(b). Each IMP is trying to send to a neighbor, but nobody has any buffers available to receive incoming packets. This situation is called **indirect store-and-forward lockup**. Note that when an IMP is locked up, all its lines are effectively blocked, including those not involved in the lockup.

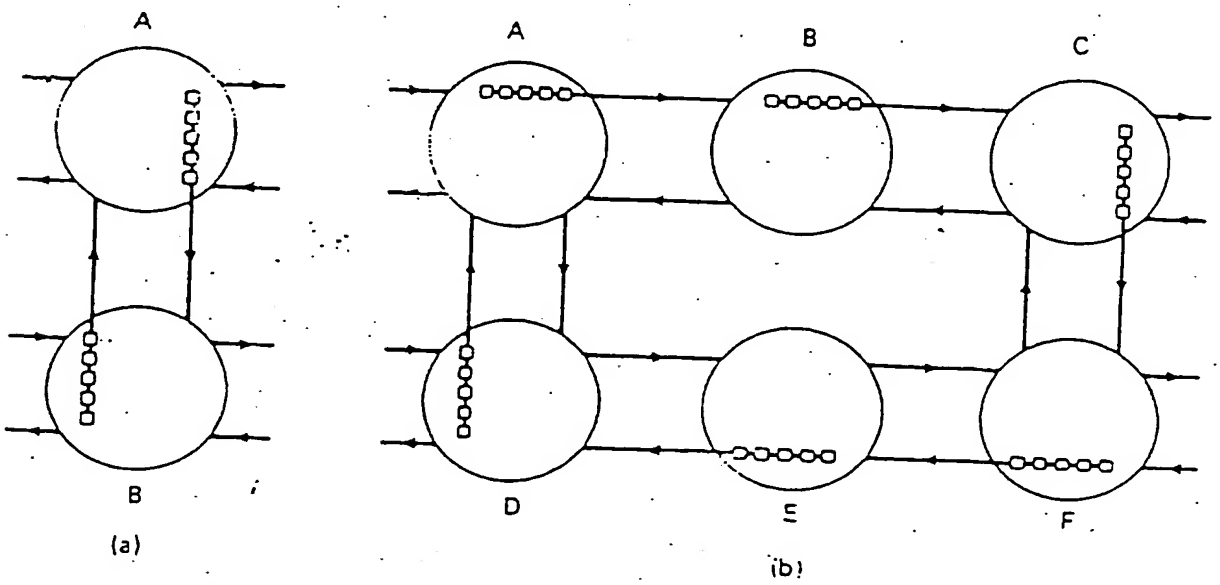

Fig. 5-23. Store-and-forward lockup. (a) Direct. (b) Indirect.

Merlin and Schweitzer (1980) have presented a solution to the problem of store-and-forward lockup. In their scheme, a directed graph is constructed, with the buffers being the nodes of the graph. Arcs connect pairs of buffers in the same IMP or adjacent IMPs. The graph is constructed in such a way that if all packets move from buffer to buffer along the arcs of the graph, then no deadlocks can occur.

As a simple example of their method, consider a subnet with N IMPs in which the longest route from any source to any destination is of length M hops. Each IMP needs $M + 1$ buffers, numbered from 0 to M. The buffer graph is now constructed by drawing an arc from buffer i in each node to buffer $i + 1$ in each of the adjacent nodes. The legal routes from buffer i at IMP A are those to a buffer labeled $i + 1$ at IMPs adjacent to A, and then to a buffer labeled $i + 2$ at IMPs two hops from A, etc.

A packet from a host can only be admitted to the subnet if buffer 0 at the source IMP is empty. Once admitted, this packet can only move to a buffer labeled 1 in an adjacent IMP, and so on, until either it reaches its destination and is removed from the subnet, or it reaches a buffer labeled M, in which case it is discarded. (If M is chosen longer than the longest route, then only looping packets will be discarded.) A packet in buffer i in some IMP may only be moved if buffer $i + 1$ in the IMP chosen by the routing algorithm is free. Note that numbering the buffers does not restrict the choice of routing algorithm, which can be static or dynamic.

To see that this algorithm is deadlock free, consider the state of all buffers labeled M at some instant. Each buffer is in one of three states: empty, holding a packet destined for a local host, or holding a packet for a distant host. In the second case the packet can be delivered, in the third case the packet is looping and must be dropped. In all three cases the buffer can be made free. Consequently, all the packets in buffers labeled $M - 1$ can now be moved forward, one at a time, to be delivered or discarded. Once all the buffers labeled $M - 1$ are free, the packets in buffers labeled $M - 2$ can be moved forward and delivered or discarded. Eventually, all packets can be delivered or discarded. If the routing algorithm guarantees that packets cannot loop, then M can be set to the longest path length, and all packets will be correctly delivered with no discards and no deadlocks.

Merlin and Schweitzer have also presented many improvements to this simple strategy to reduce the number of buffers needed and to improve the performance. For example, a packet that has already made i hops (i.e., is currently in a buffer labeled i), can be put in any available higher numbered buffer at the next hop, not just in buffer $i + 1$. As long as the sequence of buffer numbers is monotonically increasing, there can be no deadlocks.

Although this algorithm avoids deadlock completely, it has the disadvantage that under normal circumstances many buffers will be wasted due to lack of the appropriate packets. Furthermore, lines will be frequently idle because the buffer that the packet at the head of the queue happens to need is not available at the moment.

A completely different deadlock prevention algorithm that does not have these properties has been published by Blazewicz et al. (1987a). In their algorithm, each packet bears a globally unique timestamp. This stamp contains the time the packet was created in the high-order bits and the machine number in the low-order bits. It is not essential that all the clocks be synchronized, although the algorithm tends to give fairer service if the clocks are not too far apart.

The algorithm requires each IMP to reserve one buffer per input line as a

special receive buffer. All the other buffers can be used for holding packets in transit. These packets are queued in timestamp order in a separate queue for each output line, as illustrated in Fig. 5-24. In the absence of any deadlock prevention algorithm, the three IMPs in Fig. 5-24 would be deadlocked because although each one can receive a packet from its neighbors, it has nowhere to store it while the packet waits its turn for the next hop.

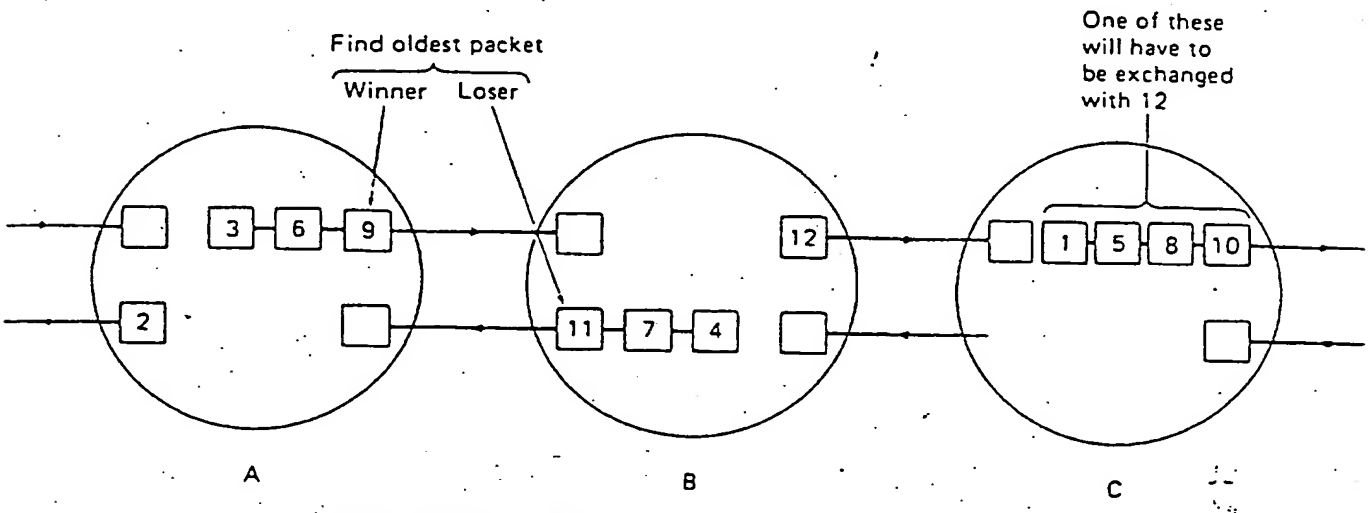

Fig. 5-24. Three potentially deadlocked IMPs.

The essence of the algorithm is that it makes a distinction between three conditions. In all three cases we assume that *A* has an important packet that it wants to send to *B*. The conditions are:

1. *B* has a free buffer.
2. *B* does not have a free buffer, but it does have a packet for *A*.
3. *B* has neither a free buffer nor a packet for *A*.

In case 1, *A* just sends its packet. In case 2, *A* and *B* exchange packets, so that the newly arrived packet can be put in the buffer just freed by the packet going the other way. In case 3, *B* is forced to choose a packet not destined for *A* (preferably, one that is at least going in the same general direction as *A*) and this packet is exchanged with *A* as in case 2.

Now we can explain the algorithm and prove that it is deadlock free. Whenever a line goes idle, the two IMPs at the ends of it exchange control packets giving the timestamp of its oldest packet that wants to use the line. The one with the lowest number (oldest packet) wins and that packet is sent, even if case 3 applies and the losing IMP is forced to send another packet in the wrong direction to free up buffer space for the winner. At any instant, one packet in the subnet has the honor of being the oldest. This packet will always come on top in any IMP-IMP discussion

out age, so this packet will make nonstop progress to its destination. As soon as it has been delivered, some other packet becomes the oldest, and it can then proceed unobstructed to its destination. As time marches on, every packet eventually becomes the oldest and is delivered.

Two criticisms can be leveled at this algorithm. First, a young packet may be sent far out of its way before it acquires enough seniority to start consistently winning the IMP-IMP timestamp comparisons. However, in practice, case 3 only occurs when the subnet is heavily loaded (all buffers full). If an IMP has, say, 3 or 4 outgoing lines and 20 to 50 full buffers, chances are that none of the outgoing queues will be empty, so that the losing IMP will nearly always be able to find a good packet to exchange with the winner.

The second criticism has to do with clock synchronization. If one IMP's clock is a few seconds ahead of all the others, its packets will be at a competitive disadvantage when timestamps are compared. Still, within a few seconds, its packets will get delivered, so the condition is annoying but not fatal. If each pair of adjacent IMPs periodically exchanges clock values, each IMP can then set its time to the average of its neighbor's times. In this way, no clock can get very far out of step with the rest.

Store-and-forward lockups are not the only kind of deadlocks that plague the subnet. Consider the situation of an IMP with 20 buffers and lines to four other IMPs, as shown in Fig. 5-25. Four of the buffers are dedicated to the four input lines, to help alleviate congestion. Assume that a sliding window protocol is being used, with window size seven. Further, assume that packets are accepted out of order by the IMP, but must be delivered to the host in order. At the time of the snapshot, four virtual circuits, 0, 1, 2, and 3, are open between the host and other hosts.

Rather than dedicate a full window load of buffers to each open virtual circuit, buffers are simply assigned on a first come, first served basis. As soon as the next sequence number expected by the host becomes available, that packet is passed to the host (with each virtual circuit independent of all the others). However higher number packets within the window are buffered in the usual way.

In Fig. 5-25 v and s indicate the virtual circuit and sequence number of each packet, respectively. The host is waiting for sequence number 0 on all four virtual circuits, but none of them have arrived undamaged yet. Nevertheless, all the buffers are occupied.

If packet 0 should arrive, it would have to be discarded due to lack of buffer space. As a result, no more packets can be passed to the host and no buffers freed. This deadlock occurred in the ARPANET in a slightly different form, and was called **reassembly lockup**.

In the ARPANET there are multipacket messages (i.e., messages too large to fit into a single packet). By allowing hosts to pass relatively large chunks of information to the IMPs in a single transfer, the number of host interrupts could be reduced. The sending IMP splits multipacket messages into individual packets and sends

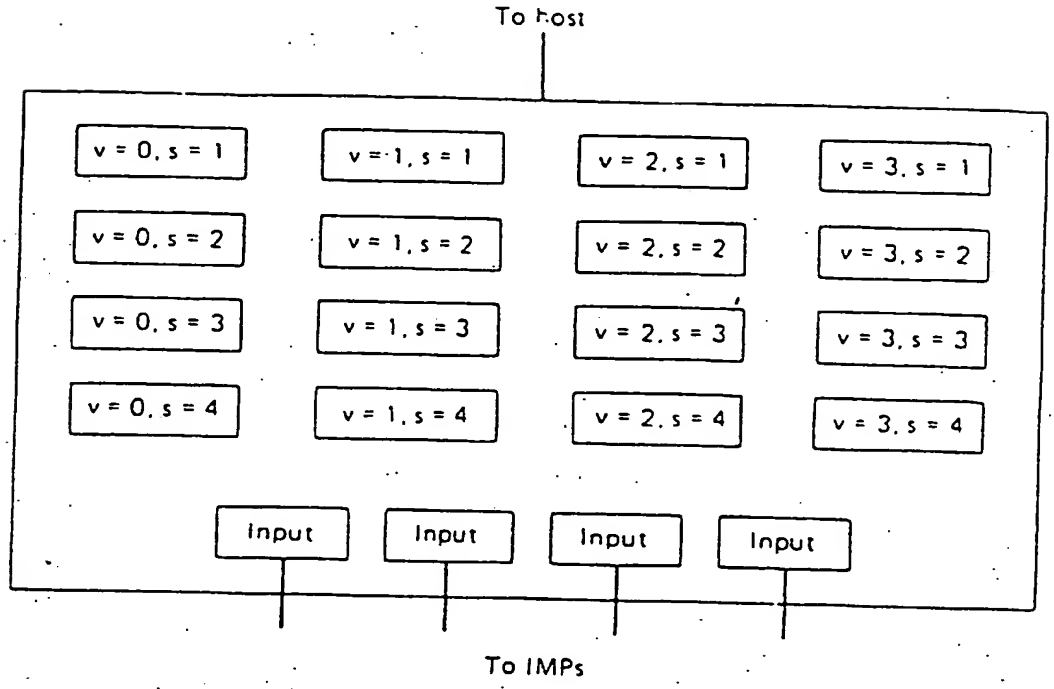

Fig. 5-25. Reassembly lockup.

each one separately. The destination IMP must put all the pieces together before passing the reassembled message to the host. If pieces of several multipacket messages have managed to commandeer all the available buffer space in the destination IMP, the missing pieces will have to be rejected and the IMP (and host) will be deadlocked.

The ARPANET solution is to have the source IMP ask the destination IMP permission to send a multipacket message. The destination IMP then dedicates enough buffers to reassemble the full message before telling the source to go ahead. The sliding window version of the deadlock can only be prevented by dedicating a full window's worth of buffers to each open virtual circuit. The reason this strategy is more attractive in the ARPANET case is that there most (96%) messages are single packet, not multipacket, whereas in the sliding window case, in effect, all messages are multipacket.

Two other problems discovered in the ARPANET are worth mentioning here. Both were caused by malfunctioning IMPs. All of a sudden, one IMP announced that it had zero delay to all other IMPs in the entire subnet. The adaptive routing algorithm spread the good news far and wide. The other IMPs were ecstatic. Within a few seconds, practically all of the traffic in the entire subnet was headed toward the only IMP that was not working properly. Although not a deadlock, this certainly brought the whole network to a grinding halt.

The other problem was also caused by a failing memory. One fine day the Aberdeen IMP (on the East Coast) decided that it was the UCLA IMP (on the West

Coast). The consequences of this case of mistaken identity can be easily imagined, especially for the UCLA and Aberdeen hosts.

The fix applied to the IMP software was to have each IMP periodically compute a software checksum of its own code and tables, in hope of discovering failing memory words, such as those that caused the above problems. Nevertheless the nagging question of how you prevent one bad IMP from bringing the whole network down remains. One of the arguments in favor of computer networking is that higher reliability can be achieved so: if one machine goes down, there are still plenty of others around. However, if a single failure at one site often pollutes the entire nationwide (or worldwide) system, there may be no advantage at all.

An exhaustive catalog of other network deadlocks and possible solutions to them is given by Lai (1982). Other deadlock prevention methods are discussed by Blazewicz et al. (1987b), and Gopal (1985).

5.4. INTERNETWORKING

Up until now, we have implicitly assumed that there is a single homogeneous network, with each machine using the same protocol in each layer. Unfortunately, this assumption is wildly optimistic. Many different networks exist. More than 20,000 SNA networks, more than 2000 DECNET networks, and uncountably many LANs of every kind imaginable are in daily operation all over the world (Green, 1986). Very few of these use the OSI model. In this section we will take a careful look at the issues that arise when it becomes necessary to interconnect two or more networks together to form an **internet**. Additional detail can be found in (Burg et al. 1984; Hinden et al. 1983; Israel and Weissberger, 1987; Postel 1980; Schneidewind 1983; Stallings 1987; and Weissberger and Israel, 1987).

Enormous controversy exists about the question of whether today's abundance of network types is a temporary condition that will go away as soon as everyone realizes how wonderful OSI is, or whether it is an inevitable, but permanent feature of the world that is here to stay. We believe that a variety of different networks will always be around, for the following reasons. First of all, the installed base of non OSI systems is already very large and growing rapidly. IBM is still selling new SNA systems. Most UNIX shops run TCP/IP. LANs are rarely OSI. This trend will continue for years because not all vendors perceive it in their interest for their customers to be able to easily migrate to another vendor's system.

Second, as computers and networks get cheaper, the place where decisions get made moves downward. Many companies have a policy to the effect that purchases costing over a million dollars have to be approved by top management, purchases costing over 100,000 dollars have to be approved by middle management, but purchases under 100,000 dollars can be made by department heads without any higher approval. This can easily lead to the accounting department installing an